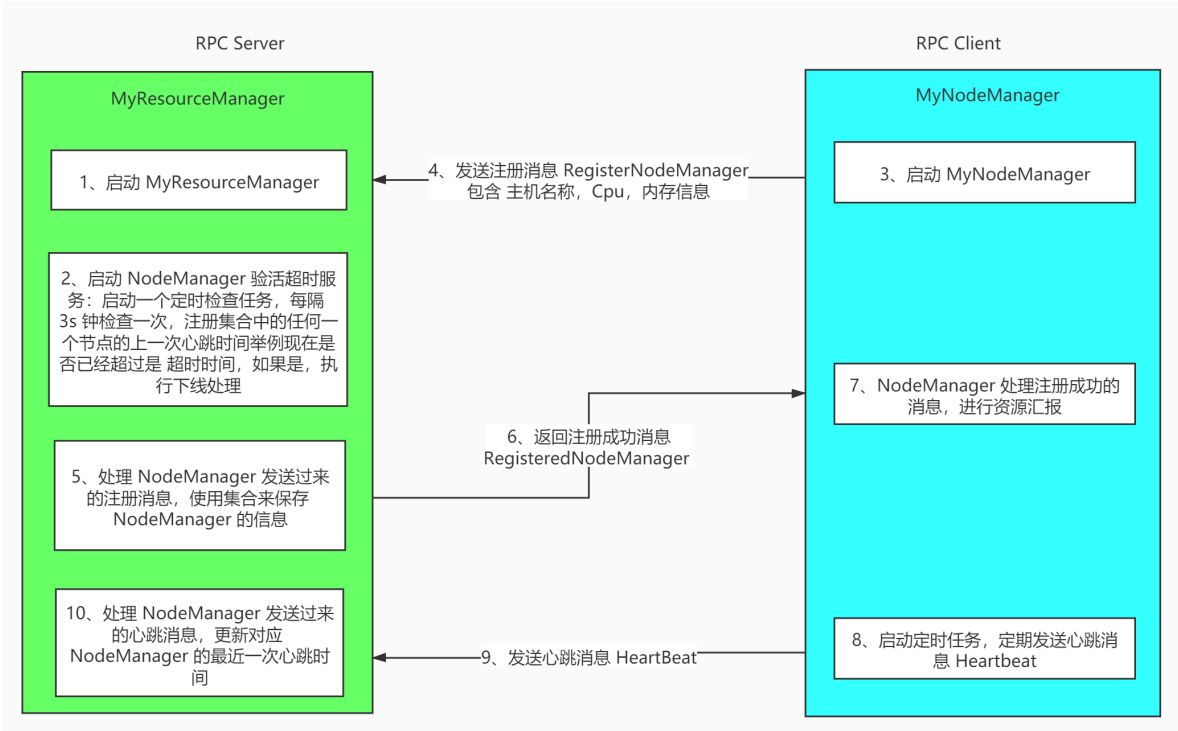
1. 使用Akka模拟实现YARN

YARN 资源管理系统/资源调度引擎， 主从架构：

```
1、ResourceManager    主节点，  管理 NodeManager
2、NodeManager        从节点，  提供资源，接收任务来执行的
```

模拟实现的需求：

```
1、ResourceManager 的实现，  NodeManager 的实现
2、ResourceManager 启动 和 NodeManager 启动
3、NodeManager 和 ResourceManager 心跳机制实现（从节点的宕机死亡的处理）
```



好几个时间：

```
1、心跳间隔时间 3
2、从节点宕机死亡最长超时时间  currentTime - lastHeatbeat > NMtimeout  = 20
3、定时服务的间隔时间  5
```

2. Akka 介绍

Akka 是一个用 Scala 编写的库，用于在 JVM 平台上简化编写具有可容错的、高可伸缩性的 Java 和 Scala 的 Actor 模型应用，其同时提供了Java 和 Scala 的开发接口。Akka 允许我们专注于满足业务需求，而不是编写初级代码。在 Akka 中，Actor 之间通信的唯一机制就是消息传递。Akka 对 Actor 模型的使用提供了一个抽象级别，使得编写正确的并发、并行和分布式系统更加容易。

重点知识：三个重要的概念： ActorSystem actor ActorRef

1、ActorSystem 管理通信角色 actor 的一个系统概念，在一个服务器节点中，只要存在一个这样的对象就可以，这个对象的作用，就是用来生成和管理所有的通信角色的生命周期

2、通信角色 Actor，存在于一台服务器中的一个 ActorSystem 的内部，用来和其他节点的 actor 进行通信。每个 Actor 都有一个 MailBox，别的 Actor 发送给它的消息都首先储存在 MailBox 中，通过这种方式可以实现异步通信。

3、每个 Actor 是单线程的处理方式，不断的从 MailBox 拉取消息执行处理，所以对于 Actor 的消息处理，不适合调用会阻塞的处理方法。

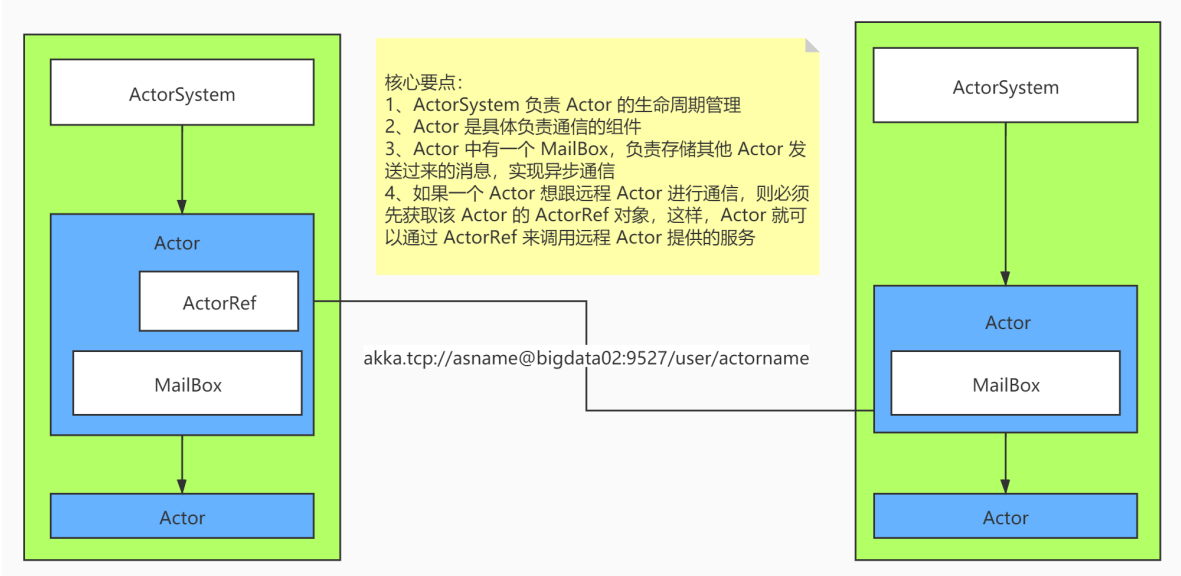
4、Actor 可以改变他自身的状态，可以接收消息，也可以发送消息，还可以生成新的 Actor

5、每一个 ActorSystem 和 Actor都在启动的时候会给定一个 name，如果要从 ActorSystem 中，获取一个 Actor，则通过以下的方式来进行 Actor 的获取：
akka.tcp://asname@bigdata02:9527/user/actorname

6、如果一个 Actor 要和另外一个 Actor 进行通信，则必须先获取对方 Actor 的 ActorRef 对象，然后通过该对象发送消息即可。

7、通过 tell 发送异步消息，不接收响应，通过 ask 发送异步消息，得到 Future 返回，通过异步回到返回处理结果。

Akka 大致工作原理图解：

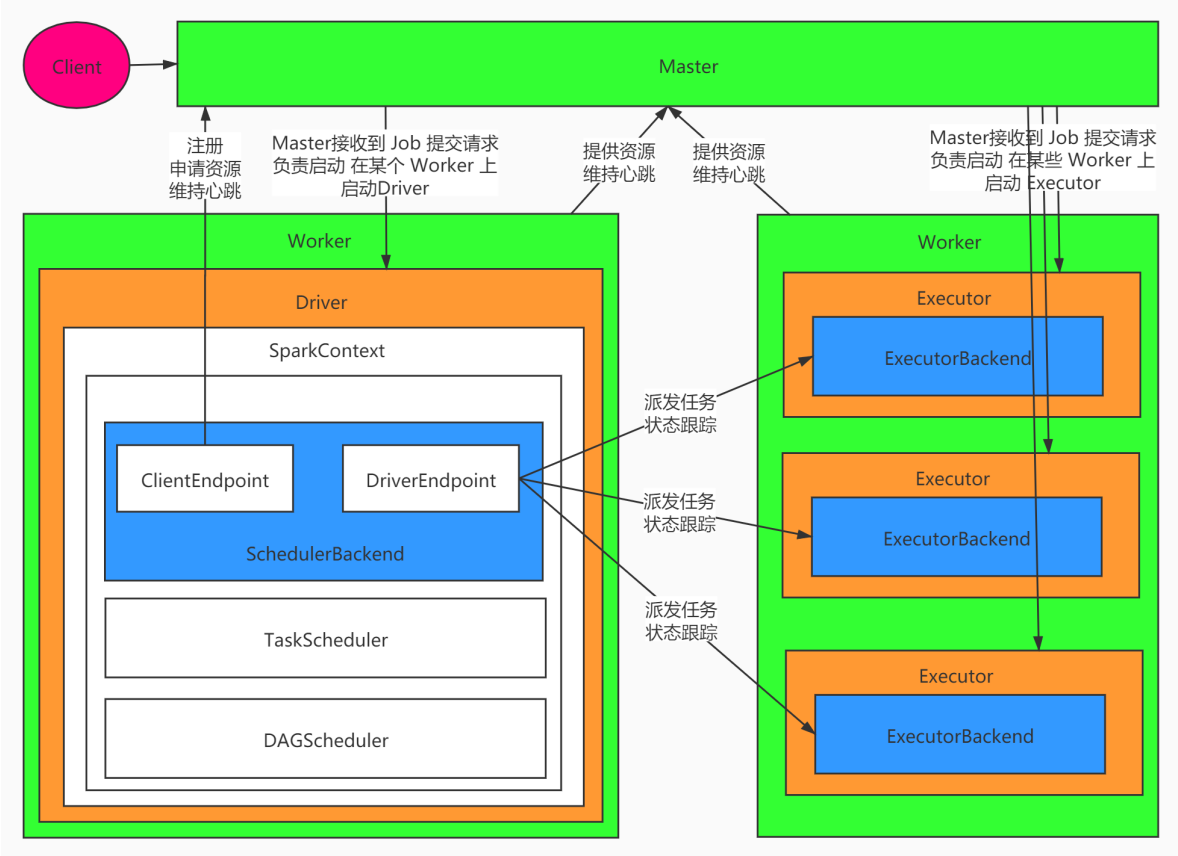


Spark 集群中 Master Worker： worker 现在要发送注册（RegisterWorker）消息给 Master: masterActorRef.send(RegisterWorker)

Spark-1.x 版本中的应用程序执行的时候，会生辰给一个 Driver 和 多个 Executor 。

它的内部就有两个 Actor：

```
1、DriverActor： 负责发送任务给其他的 worker 中的 executor 来执行的，作用和 Spark-2.x 版本中的 DriverEndpoint 是一样的
2、ClientActor： 负责和 master 进行通信，作用和 Spark-2.x 版本中的 ClientEndpoint 是一样的
```



3. 模拟实现 YARN 代码实现

3.1. MyResourceanager 代码实现

具体代码实现：

```
package com.mazh.scala.core.day4.rpc.yarn01

import akka.actor.{Actor, ActorSystem, Props}
import com.typesafe.config.ConfigFactory

import scala.collection.mutable

/**************************************************
 * TODO_MA 马中华 https://blog.csdn.net/zhongqi2513
 *  注释： 集群主节点抽象
 */
```

```scala
class MyResourceManager(var hostname: String, var port: Int) extends Actor {

    // TODO_MA 注释： 用来存储每个注册的NodeManager节点的信息
    private var id2nodemanagerinfo = new mutable.HashMap[String,
NodeManagerInfo]()

    // TODO_MA 注释： 对所有注册的NodeManager进行去重，其实就是一个HashSet
    private var nodemanagerInfoes = new mutable.HashSet[NodeManagerInfo]()

    // TODO_MA 注释： actor在最开始的时候，会执行一次
    override def preStart(): Unit = {
        import scala.concurrent.duration._
        import context.dispatcher

        // TODO_MA 注释： 调度一个任务， 每隔五秒钟执行一次
        context.system.scheduler.schedule(0 millis, 5000 millis, self,
CheckTimeOut)
    }

    // TODO_MA 注释： 正经服务方法
    override def receive: Receive = {

        case RegisterNodeManager(nodemanagerid, memory, cpu) => {
            val nodeManagerInfo = new NodeManagerInfo(nodemanagerid, memory,
cpu)
            println(s"节点 ${nodemanagerid} 上线")

            // TODO_MA 注释： 对注册的NodeManager节点进行存储管理
            id2nodemanagerinfo.put(nodemanagerid, nodeManagerInfo)
            nodemanagerInfoes += nodeManagerInfo

            // TODO_MA 注释： 把信息存到zookeeper
            sender() ! RegisteredNodeManager(hostname + ":" + port)
        }

        case Heartbeat(nodemanagerid) => {
            val currentTime = System.currentTimeMillis()
            val nodeManagerInfo = id2nodemanagerinfo(nodemanagerid)
            nodeManagerInfo.lastHeartBeatTime = currentTime

            id2nodemanagerinfo(nodemanagerid) = nodeManagerInfo
            nodemanagerInfoes += nodeManagerInfo
        }

        // TODO_MA 注释： 检查过期失效的 NodeManager
        case CheckTimeOut => {
            val currentTime = System.currentTimeMillis()

            // TODO_MA 注释： 15 秒钟失效
            nodemanagerInfoes.filter(nm => {
                val heartbeatTimeout = 15000
                val bool = currentTime - nm.lastHeartBeatTime > heartbeatTimeout
                if (bool) {
                    println(s"节点 ${nm.nodemanagerid} 下线")
                }
                bool
            }).foreach(deadnm => {
                nodemanagerInfoes -= deadnm
```

```
                id2nodemanagerinfo.remove(deadnm.nodemanagerid)
            })
            println("当前注册成功的节点数" + nodemanagerInfoes.size + "\t分别是：" +
nodemanagerInfoes.map(x => x.toString)
              .mkString(","));
        }
    }
}

// TODO_MA 注释：  启动入口
object MyResourceManager {
    def main(args: Array[String]): Unit = {
        val str =
        s"""
           |akka.actor.provider = "akka.remote.RemoteActorRefProvider"
           |akka.remote.netty.tcp.hostname = localhost
           |akka.remote.netty.tcp.port = 6789
      """.stripMargin

        val conf = ConfigFactory.parseString(str)

        // TODO_MA 注释：ActorSystem
        val actorSystem = ActorSystem(Constant.RMAS, conf)

        // TODO_MA 注释：启动了一个actor ：  MyResourceManager
        actorSystem.actorOf(Props(new MyResourceManager("localhost", 6789)),
Constant.RMA)
    }
}
```

3.2. MyNodeManager 代码实现

具体代码实现：

```
package com.mazh.scala.core.day4.rpc.yarn01

import akka.actor.{Actor, ActorSelection, ActorSystem, Props}
import com.typesafe.config.ConfigFactory

/** ********************************************
 * TODO_MA 马中华 https://blog.csdn.net/zhongqi2513
 *  注释：
 *  1、spark rpc 生命周期方法：  onStart receive onStop
 *  2、akka  rpc 生命周期方法：   preStart  receive postStop() */
class MyNodeManager(val nmhostname: String, val resourcemanagerhostname: String,
val resourcemanagerport: Int, val memory: Int,
    val cpu: Int) extends Actor {

    var nodemanagerid: String = nmhostname
    var rmRef: ActorSelection = _

    // TODO_MA 注释：  会提前执行一次
    override def preStart(): Unit = {
```

```
        // 远程path                          akka.tcp://（ActorSystem的名称）@（远程地址
的IP）    :           （远程地址的端口）/user/（Actor的名称）
        rmRef = context.actorSelection(s"akka.tcp://${
            Constant.RMAS

 }@${resourcemanagerhostname}:${resourcemanagerport}/user/${Constant.RMA}")

        println(nodemanagerid + " 正在注册")
        rmRef ! RegisterNodeManager(nodemanagerid, memory, cpu)
    }

    // TODO_MA 注释：  正常服务方法
    override def receive: Receive = {
        case RegisteredNodeManager(masterURL) => {
            println(masterURL);

            // TODO_MA 注释：  initialDelay: FiniteDuration, 多久以后开始执行
            // TODO_MA 注释：  interval:     FiniteDuration, 每隔多长时间执行一次
            // TODO_MA 注释：  receiver:     ActorRef, 给谁发送这个消息
            // TODO_MA 注释：  message:      Any  发送的消息是啥
            import scala.concurrent.duration._
            import context.dispatcher
            context.system.scheduler.schedule(0 millis, 4000 millis, self,
SendMessage)
        }
        case SendMessage => {
            // TODO_MA 注释：  向主节点发送心跳信息
            rmRef ! Heartbeat(nodemanagerid)

            println(Thread.currentThread().getId)
        }
    }
}

// TODO_MA 注释：  启动类
object MyNodeManager {
    def main(args: Array[String]): Unit = {

        // TODO_MA 注释：  远程主机名称
        val HOSTNAME = args(0)

        // TODO_MA 注释：   RM 的 hostname 和 port
        val RM_HOSTNAME = args(1)
        val RM_PORT = args(2).toInt

        // TODO_MA 注释：   抽象的内存资源 和 CPU 个数
        val NODEMANAGER_MEMORY = args(3).toInt
        val NODEMANAGER_CORE = args(4).toInt

        // TODO_MA 注释：   当前 NM 的 hostname 和 port
        var NODEMANAGER_PORT = args(5).toInt
        var NMHOSTNAME = args(6)
        val str =
            s"""
               |akka.actor.provider = "akka.remote.RemoteActorRefProvider"
               |akka.remote.netty.tcp.hostname = ${HOSTNAME}
               |akka.remote.netty.tcp.port = ${NODEMANAGER_PORT}
            """.stripMargin
```

```
        val conf = ConfigFactory.parseString(str)
        val actorSystem = ActorSystem(Constant.NMAS, conf)
        actorSystem.actorOf(Props(new MyNodeManager(NMHOSTNAME, RM_HOSTNAME,
RM_PORT, NODEMANAGER_MEMORY, NODEMANAGER_CORE)), Constant.NMA)
    }
}
```

3.3. 辅助类 Message 实现

辅助类 Message 具体代码实现：

```
package com.mazh.scala.core.day4.rpc.yarn01

// TODO_MA 注释： 注册消息   nodemanager  -> resourcemanager
case class RegisterNodeManager(val nodemanagerid: String, val memory: Int, val
cpu: Int)

// TODO_MA 注释： 注册完成消息 resourcemanager -> nodemanager
case class RegisteredNodeManager(val resourcemanagerhostname: String)

// TODO_MA 注释： 心跳消息  nodemanager -> resourcemanager
case class Heartbeat(val nodemanagerid: String)

// TODO_MA 注释： NodeManager 信息类
class NodeManagerInfo(val nodemanagerid: String, val memory: Int, val cpu: Int)
{

    // TODO_MA 注释： 上一次心跳时间
    var lastHeartBeatTime: Long = _

    override def toString: String = {
        nodemanagerid + "," + memory + "," + cpu
    }
}

// TODO_MA 注释： 一个发送心跳的信号
case object SendMessage

// TODO_MA 注释： 一个检查信号
case object CheckTimeOut
```

3.4. 辅助类 Constant 实现

辅助类 Constant 具体代码实现：

```scala
package com.mazh.scala.core.day4.rpc.yarn01

// TODO_MA 注释：  一些信息类
object Constant {
    val RMAS = "MyResourceManagerActorSystem"
    val RMA = "MyResourceManagerActor"
    val NMAS = "MyNodeManagerActorSystem"
    val NMA = "MyNodeManagerActor"
}
```

4. 运行测试

1、先运行 MyResourceManager。

2、再运行 MyNodeManager， 需要指定程序的一些参数，具体看注释和演示。